TEKNILLINEN TIEDEKUNTA

OPENCV -KONENÄKÖKIRJASTO JA VIRHEIDEN TUNNISTAMINEN TERÄSPINNOISTA

Eemil Kiviahde

KONETEKNIIKAN TUTKINTO-OHJELMA

Kandidaatintyö

Heinäkuu 2021

TIIVISTELMÄ

OpenCV -konenäkökirjasto ja virheiden tunnistaminen teräspinnoista

Eemil Kiviahde

Kandidaatintyö 2021, 33 s.

Työn ohjaaja yliopistolla: Yrjö Louhisalmi

Tässä kandidaatintyössä perehdytään virheiden tunnistamiseen teräsnauhojen sekä valuaihioiden pinnoista. Työssä käydään läpi konenäön perusmenetelmiä sekä toteutustapoja ja näiden toteutusta OpenCV konenäkökirjaston avulla. Konkreettisia esimerkkejä lasketaan hyödyntäen todellista kuvadataa terästeollisuudesta. Myös syväoppimista sivutaan. Työssä nähdään, ettei virheiden tunnistus teräspinnoista konenäön avulla ole helpoin mahdollinen tehtävä. Konenäön avulla on mahdollista kerätä suuria määriä tietoa prosessien eri vaiheista ja tiedosta hyötyminen voi olla haastavaa, mutta oikein toteutettuna konenäkö tuottaa arvoa sekä voi vähentää resurssien turhaa kulutusta.

Asiasanat: konenäkö, koneäly, koneoppiminen, OpenCV

ABSTRACT

The OpenCV machine vision library and detection of defects in steel surfaces

Eemil Kiviahde

Bachelor's thesis 2021, 33 p.

Supervisor at the university: Yrjö Louhisalmi

The aim of this thesis is to familiarize the reader in the detection of defects in steel surfaces. The basic machine vision methods and their implementation using the OpenCV machine vision library are reviewed. Concrete examples are calculated utilizing real image data from the steel industry. Deep learning is also touched on. It becomes apparent that the detection of defects from steel surfaces using machine vision is not the easiest task. Machine vision makes it possible to gather large amounts of information about various process stages. Utilizing this information may be challenging, but when implemented correctly it can produce value and reduce unnecessary consumption of resources.

Keywords: machine vision, machine learning, OpenCV

ALKUSANAT

Tämän kandidaatintyön tarkoituksena oli perehtyä konenäön perusmenetelmiin ja rakenteisiin, sekä OpenCV -konenäkökirjastoon vaihtoehtona näiden menetelmien toteuttamisessa.

Innoituksena kandidaatintyölleni toimi kesätyöni Sapotech Oy:ssä. Kesän aikana pääsin oppimaan paljon konenäöstä ja virheiden tunnistamisesta, ja haluan kiittää toimitusjohtaja Saku Kaukosta inspiraatiosta tämän työn aiheeseen sekä opetuksesta ja kaikesta tiedosta aiheeseen liittyen.

Haluan myös kiittää Yrjö Louhisalmea työn ohjauksesta.

Oulu, 2.7.2021

Eemil Kiviahde

SISÄLLYSLUETTELO

MERKINNÄT JA LYHENTEET

ASM	Angular Second Moment
GLCM	Grey Level Co-Occurrence Matrix
KNN	K-Nearest Neighbor
LBP	Local Binary Pattern
OpenCV	Open Computer Vision
ORB	Oriented FAST and Rotated BRIEF
RGB	Red Green Blue
SIFT	Scale Invariant Feature Transform
SVM	Support Vector Machine
ToF	Time-of-Flight

1 JOHDANTO

Konenäöllä tarkoitetaan digitaalisen kuvan prosessointia tietokoneellisesti niin, että kuvasta saadaan selville halutun kaltaista informaatiota. Tätä informaatiota voidaan hyödyntää joko tietokoneen suorittamissa jatkoprosessoinneissa tai vaikkapa jonkin tietokoneen ulkoisen tai tietokoneesta erillisen toiminnon ohjaamiseen.

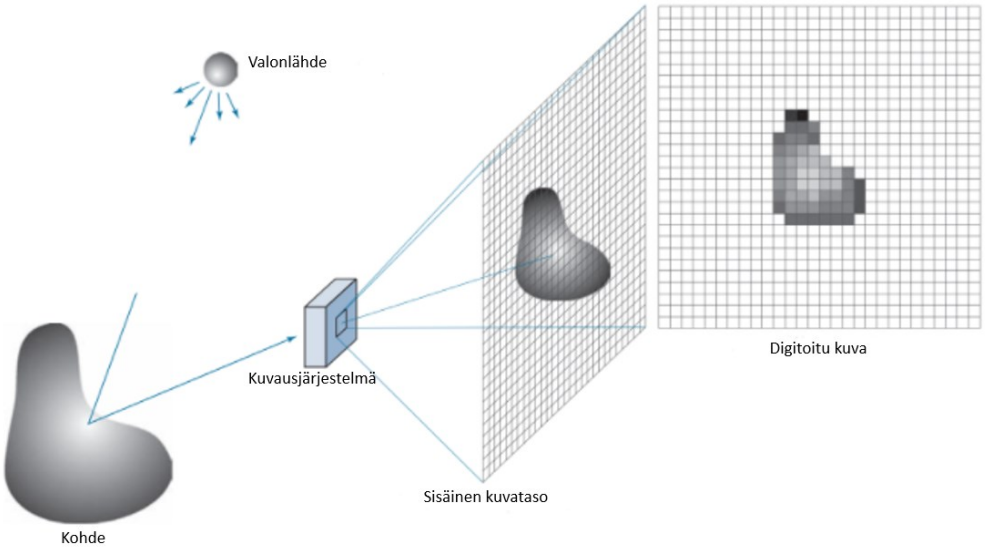
Tässä työssä haluttuna informaationa toimii tieto teräspinnan sisältämistä mahdollisista virheistä. Riippuen, onko kyseessä valuaihio tai esimerkiksi teräsnauha eri prosessivaiheissaan, voidaan tätä informaatiota hyödyntää päätettäessä terästuotteen jatkokäsittelystä.

1.1 Digitaalinen kuva ja kuvankäsittely

Kuva voidaan esittää funktiona $f(x, y)$, jonka arvo eri koordinaateissa x ja y kuvastaa intensiteettiä. Kun koordinaatit sekä intensiteetin arvot ovat äärellisiä sekä diskreettejä, kyseessä on digitaalinen kuva. Digitaalinen kuva koostuu siis elementeistä, joilla on sijainti sekä arvo. Näitä elementtejä kutsutaan yleisesti pikseleiksi. (Gonzalez & Woods 2018, s. 18)

1.1.1 Kuvan muodostaminen

Digitaalisen kuvan muodostamisessa kuvausjärjestelmä kerää kohteesta heijastuvan valon ja kohdistaa sen kuvaustasoon, valon tapauksessa tämä tapahtuu optisella linssillä (Kuva 1). Kuvaustasossa sijaitseva sensori tuottaa kerätyn valon määrään suhteutetun ulostulosignaalin. Tämä ulostulosignaali muunnetaan digitaaliseksi kuvaksi. (Gonzalez & Woods 2018, s. 61)



Kuva 1. Esimerkki kuvan muodostamisesta (mukaillen Gonzalez & Woods 2018).

Kuvaa esittävä funktio $f(x, y)$ voidaan esittää matriisimuodossa yhtälön (1) tavoin:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \quad (1)$$

missä M on kuvan korkeussuuntaisten elementtien määrä
N on kuvan leveyssuuntaisten elementtien määrä

Matriisin (1) jokainen elementti on pikseli. Pikselin arvoa kuvaavaa intensiteettiä ja sen tarkkuuttaa joudutaan rajaamaan tietokoneiden teknisten rajoitteiden vuoksi. Intensiteettitasojen määrä saadaan yhtälöstä (2):

$$L = 2^k \quad (2)$$

missä L on mahdollisten intensiteettitasojen määrä
k on yhdelle pikselille varattu tila biteissä, ns. ”bittisyvyys”

Esimerkiksi yhtälön (2) mukaan kuvan, jonka bittisyvyys on 8, yksi pikseli voi esittää $2^8 = 256$ eriä intensiteettiä. Koska tiedetään intensiteetin olevan kerätyn valon määrään suhteutettu luku, on se aina positiivinen, ellei toisin sovita. Indeksoinnista johtuen 8 bitin syvyiselle kuvalle intensiteettiarvot voivat siis olla välillä [0,255].

Värikuvan tapauksessa pikseli voi sisältää kuvaformaatista riippuen useita intensiteettiarvoja, jotka voivat kuvata värejä, sävyä, kirkkautta tai värikylläisyyttä. RGB-kuvan tapauksessa pikseli sisältää intensiteettiarvot punaiselle (Red), vihreälle (Green) ja siniselle (Blue). Pikselin sisältämien arvojen määrää kuvataan niin sanottujen kanavien määrällä. RGB-kuvan tapauksessa värikanavia on 3. Pikseli voi sisältää myös esimerkiksi valonnopeuden ja kulkuajan avulla lasketun syvyys- tai etäisyysarvon (Time-of-Flight-menetelmä, ToF). Teräspinnat ovat kuitenkin usein melko värittömiä, joten niiden käsittelyssä harmaasävykuvat (Kuva 2) ovat riittäviä. Tällä voidaan myös vähentää muistin tarvetta sekä yleistä prosessoinnin määrää. (Davies 2005, s. 21–23)

Kuva 2. Esimerkki värikuvasta ja vastaavasta harmaasävykuvasta.

1.1.2 Kuvankäsittelyn perusfunktioita

Kun tiedetään kuvan koostuvan lukuisista intensiteettiarvoista, voidaan arvoja tarkoituksellisesti muuntamalla muuttaa kuvan luonnetta sekä mahdollisesti helpottaa sen myöhempää koneellista käsittelyä. Kuvan kirkkautta voidaan esimerkiksi muuntaa lisäämällä kaikkiin pikseleihin jokin vakio yhtälön (3) mukaan (mukaillen Davies 2005, s. 31):

$$Q = P + b \tag{3}$$

missä Q on uusi intensiteetin arvo
P on alkuperäinen intensiteetti
b on mielivaltainen vakio

On muistettava, ettei intensiteetti voi alittaa nollaa, ja että kuvan bittisyvyys määrittelee intensiteetin maksimiarvon. Jos siis $P + b < 0$, on uuden intensiteetin arvoksi tultava 0, kun taas summan ollessa suurempaa kuin bittisyvyyden määrittelemä intensiteetin maksimiarvo, on uuden intensiteetin arvoksi tultava tuo maksimiarvo. Yhtälö (3) toimii, sillä intensiteettiarvo kuvastaa suoraan kirkkautta. (Davies 2005 s. 31)

Kuvan kontrastia voidaan taas muuntaa kertomalla kuvan kaikkia pikseleitä jollakin vakiolla yhtälön (4) mukaan (mukaillen Davies 2005, s. 31):

$$Q = P * g \tag{4}$$

missä g on mielivaltainen vakio

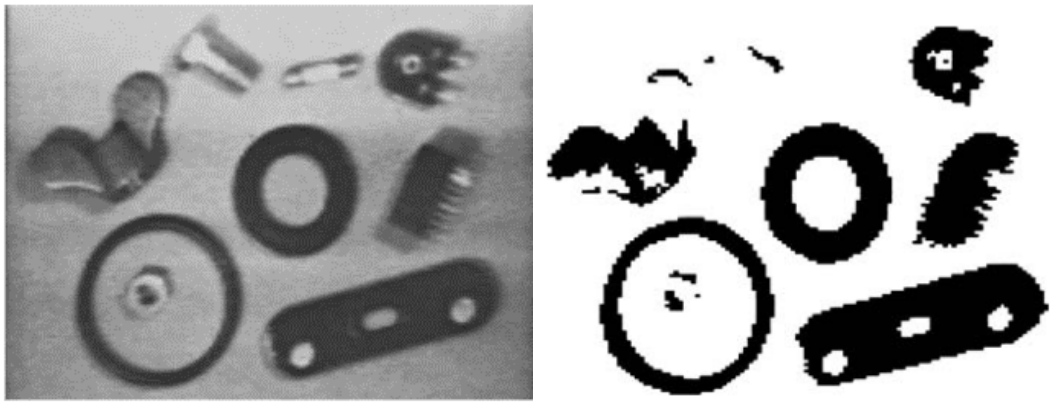
Yhtälö (4) toimii, sillä suuren intensiteettiarvon tulo on aina suurempi kuin pienen intensiteettiarvon tulo, jolloin arvojen välinen kontrasti kasvaa.

Kynnysoperaatiossa (threshold) asetetaan haluttu intensiteettiarvoväli, jolla olevat pikselit halutaan säilyttää. Pikselit, jotka eivät kuulu määritellylle välille, muutetaan esimerkiksi mustiksi ja välille kuuluvat pikselit valkoisiksi. Operaation logiikka 8 bitin syvyiselle kuvalle noudattaa yhtälöä (5) (mukaillen Davies 2005, s. 31):

$$\begin{cases} Q = 255, P \geq t \\ Q = 0, P < t \end{cases} \quad (5)$$

missä t on mielivaltainen vakio välillä [0,255]

Kynnysoperaation tuloksena saadaan periaatteessa binäärimuotoinen kuva, jossa pikselit ovat joko päällä (= 1) tai pois päältä (= 0) (Kuva 3).



Kuva 3. Esimerkki kuvasta, jolle on tehty kynnysoperaatio (Davies 2005).

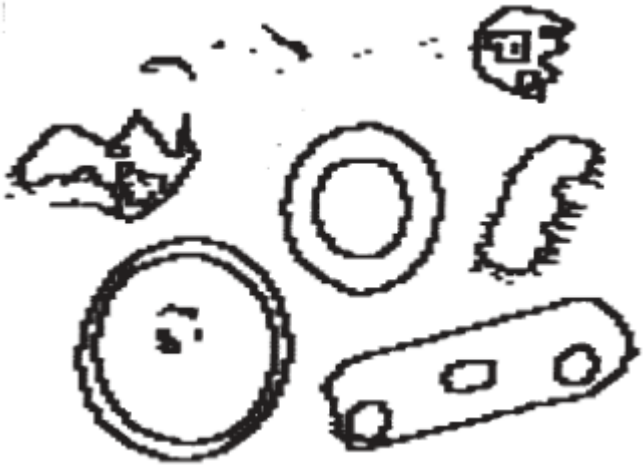
Reunantunnistus (Kuva 4) on monelle tunnistusoperaatiolle hyödyllinen toimenpide. Kuvasta voidaan tunnistaa reunoja monella tavalla, mutta kuvalle, jolle on suoritettu kynnysoperaatio, voidaan reunantunnistus suorittaa esimerkiksi yhtälön (6) mukaisesti (mukaillen Davies 2005, s. 34):

$$\begin{cases} Q = P, P0 > P = P1 \\ Q = P, P1 > P = P0 \\ Q = 0, P1 = P = P0 \end{cases} \quad (6)$$

missä P on tarkasteltavan pikselin intensiteettiarvo

P0 on tarkasteltavaa edeltävän pikselin intensiteettiarvo

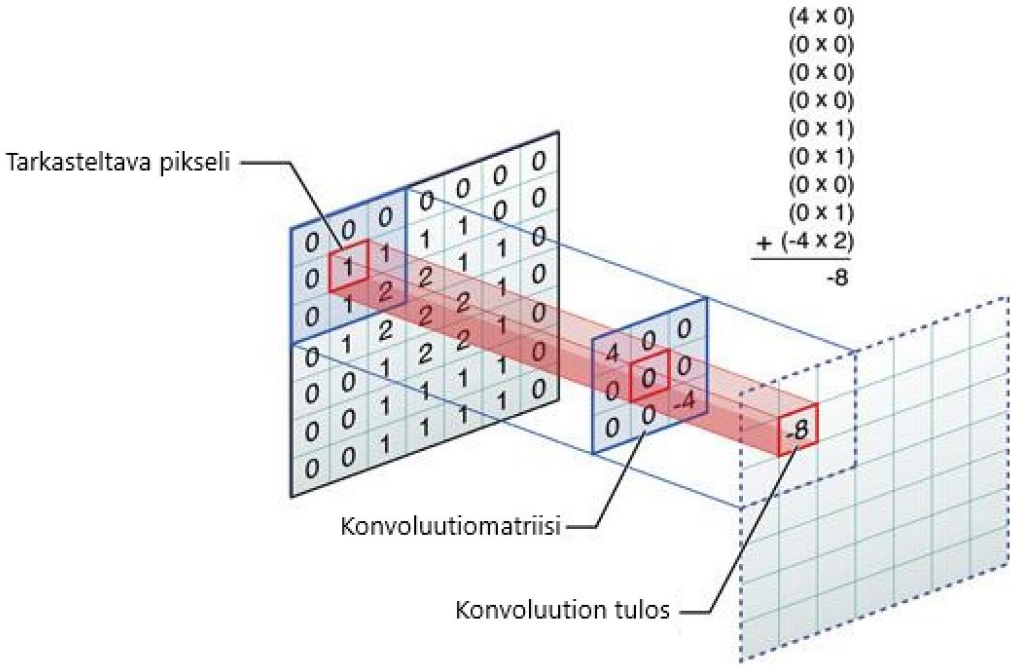
P2 on tarkasteltavan jälkeisen pikselin intensiteettiarvo



Kuva 4. Esimerkki reunantunnistuksen tuloksesta (Davies 2005).

1.1.3 Konvoluutiot ja Gaussin tasoitusmenetelmä

Kuvia voidaan käsitellä konvoluutiomatriiseja, ns. ”kerneleitä” tai ”maskeja” hyödyntäen. Konvoluutiossa konvoluutiomatriisi konseptuaalisesti ”liu-utetaan” kuvassa tarkasteltavan pikselin päälle ja matriisin arvoilla sekä tarkastelupikselin ja sen ympäristön pikseleiden arvoilla suoritetaan haluttu operaatio. (Kuva 5). Lopputuloksena saadaan uusi, muunneltu kuva. (Shapiro & Stockman 2001, s. 64–66)



Kuva 5. Esimerkki konvoluutiosta.

Konvoluutiomatriisin muotoa ja arvoja muuttamalla voidaan toteuttaa erilaisia operaatioita, kuten kuvan tasoitusta ja terävöitystä.

Kuvaa tasoittava konvoluutiomatriisi voidaan luoda yhtälön (7) mukaisella Gaussin funktiolla origon ollessa matriisin keskellä:

$$ce^{-\frac{x^2+y^2}{2*\sigma^2}} \tag{7}$$

missä c on mielivaltainen vakio
x on konvoluutiomatriisin vaakakoordinaatti
y on konvoluutiomatriisin pystykoordinaatti
σ on keskihajonta

Vakio c valitaan niin, että kaikki matriisin elementit ovat kokonaislukuja. Esimerkiksi Gaussin mukainen 7x7 konvoluutiomatriisi saadaan arvoilla c = 90 ja $\sigma^2 = 2$ (Kuva 6). Konvoluutiossa tarkastelupikselin arvoksi asetetaan konvoluutiomatriisin arvojen perusteella painotettu keskiarvo. (Shapiro & Stockman 2001, s. 166–169)

1	3	7	9	7	3	1
3	12	26	33	26	12	3
7	26	55	70	55	26	7
9	33	70	90	70	33	9
7	26	55	70	55	26	7
3	12	26	33	26	12	3
1	3	7	9	7	3	1

Kuva 6. Gaussin mukainen 7x7 konvoluutiomatriisi.

1.2 Laskentatehon kehitys

Mooren (1965) mukaan transistorien määrä integroiduissa piireissä tuplaantuu vuosittain. Vaikkakaan laki ei ole fyysisesti definitiivinen vaan pikemminkin empiirinen havainto, se kuvaa hyvin teknologian kehityksen nopeutta. Mooren tekemä ennuste oli, että tuo kehitys jatkuu vuoteen 1975. Voidaan kiistatta sanoa, että vuodesta 1975 laskentateho on kasvanut merkittävästi. Samalla myös digitaaliset kuvat ovat kehittyneet kameroiden kehittyessä, ja kuvien koko on kasvanut merkittävästi. Kuvakoon kasvu vaikuttaa suoraan kuvaprosessoinnissa vaadittavaan tehoon sekä muistikapasiteettiin. Voidaan tosin todeta, että laskentatehon kehitys on mennyt pitkästi ohi näiden kehittyneiden kuvien perusprosessoinnista. Laskentateho mahdollistaakin nykyään todella raskaita prosesseja, kuten kuvantunnistuksen syväoppimisen avulla. Lisäksi kehitys on mahdollistanut reaaliaikaisen kuvien prosessoinnin.

1.2.1 Massadata terästeollisuudessa

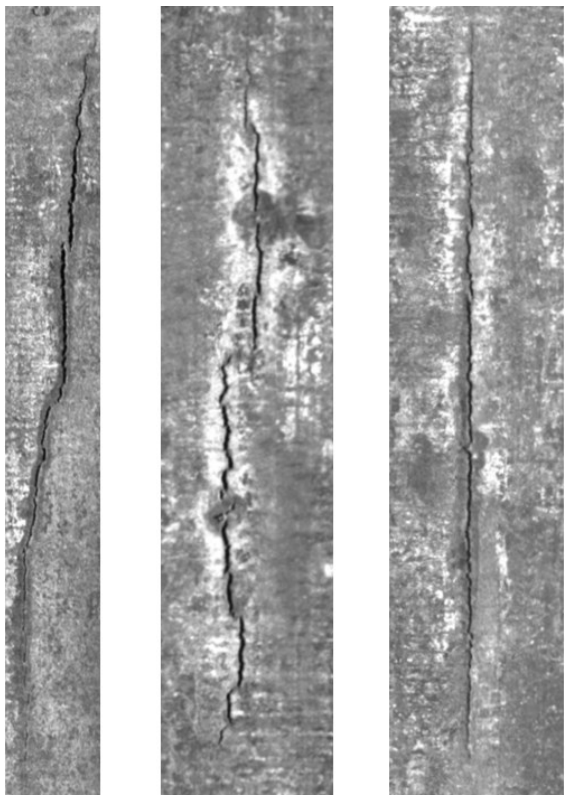
Laskentatehon ja muistikapasiteettien kehitys on mahdollistanut myös massiivisten tietomäärien prosessoinnin ja esimerkiksi syväoppimisen tarpeisiin suuren tietomäärän keräämisen ja siitä hyötymisen. Syväoppimisessa tietomäärä korreloi suoraan mallin tarkkuuden kanssa. Terästeollisuuden kokoluokan takia on teräksen eri valmistusvaiheista mahdollista kerätä suuria määriä tietoa. Ongelmaksi voikin muodostua tuon tiedon järkevä hyödyntäminen. On helppoa myös päätyä keräämään turhaa tietoa, joka ei hyödytä prosessia.

Laadunhallinnassa suurista tietomääristä on hyötyä, sillä niiden pohjalta voidaan luoda tarkkoja historiatietoja sekä kartoittaa tarkasti tuotteen elinkaaren alkupää. Parhaimmillaan tietoa voidaan kerätä jo raakamateriaalien hankinnasta ja tietoketju voi pysyä katkeamattomana puolivalmisteen valmistumiseen asti. Jos asiakkaan saamassa tuotteessa ilmeneekin virhe, voidaan sen alkuperä parhaimmillaan kartoittaa näiden tietojen pohjalta tarkasti. Ideaalitilanteessa toki tätä massiivista tietomäärää hyödynnettäisiin estämään virheiden synty täysin, mutta prosessien monimutkaisuuden takia tämä on hankalaa. Prosessien toimintaa ei välttämättä pystytä ymmärtämään tai ohjaamaan niin tarkasti, että kerättyä tietoa pystyttäisiin hyödyntämään virheiden kokonaisvaltaisessa poistamisessa.

Kuvantunnistuksen keinoin on kuitenkin mahdollista estää selvien virheiden pääsy loppukäyttäjälle, ja oppivia järjestelmiä käyttämällä virheistäkin voidaan hyötyä. Tunnistusalgoritmia voidaan aina uusien virhetyyppien ilmetessä opettaa tunnistamaan niitä. Oppivan järjestelmän tapauksessa tämä on helpompaa, kuin perinteisten algoritmien kanssa, jotka vaativat koodin muuntamista tai parametrien säätöä.

1.3 Etsittävät virheet teräspinnoista

Teräspinnoista halutaan löytää virheitä eri tuotannon vaiheissa tuotteen laadun sekä prosessin eri vaiheiden toiminnan varmistamiseksi. Virhe teräsnauhan pinnassa voi olla merkki linjan toimintahäiriöstä, tai se voi vaikuttaa tuotteelle suoritettaviin jatkotoimenpiteisiin. Kun esimerkiksi valuaihion virheiden olemassaolo ja luonne tiedetään, voidaan tietoa hyödyntää aihion hionnassa. Esimerkkejä konenäöllä etsittävistä virheistä ovat halkeamat, naarmut, painaumat, repeämät ja tartuntajäljet teräsnauhoissa ja aihioissa (Kuva 7). Virheiden tunnistuksessa pinnan hyvä valaiseminen ja riittävän terävän ja tarkan kuvan ottaminen ovat erittäin olennaisessa osassa, sillä osa virheistä voi olla pienikokoisia ja hankalia huomata jopa silmällä. Tunnistusta hankaloittaa myös pintojen ulkonäön vaihtelevuus teräslajeittain.



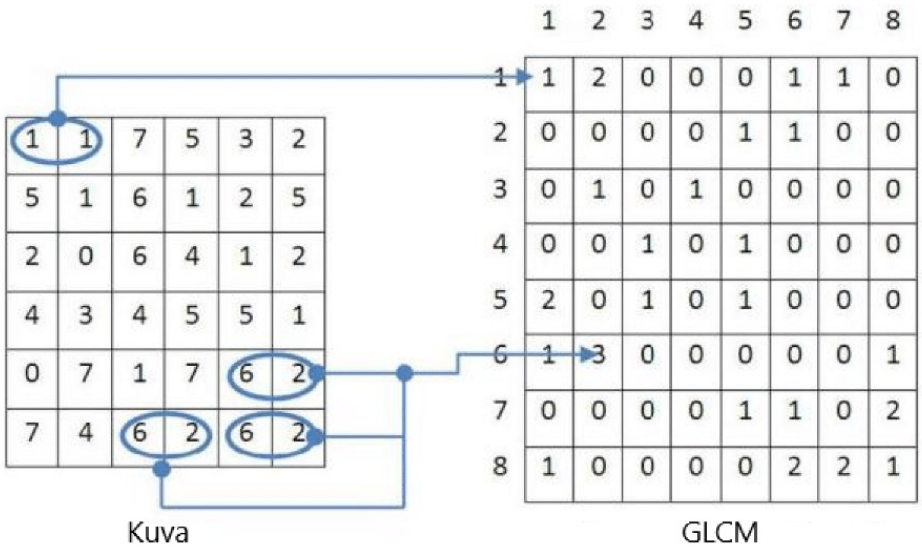
Kuva 7. Esimerkkejä etsittävistä halkeamista (Sapotech Oy).

2 VIRHEEN TUNNISTAMISEN TUNNUSLUVUT

Digitaalisesta kuvasta voidaan eri menetelmin laskea kuvan tekstuuria sekä kuvan eri elementtejä kuvaavia lukuja ja matriiseja. Kuvantunnistuksessa on hyödyllistä löytää tunnusluku, joka kuvaa etsittävää piirrettä mahdollisimman hyvin myös kuvan kontrastin, kirkkauden, orientaation ja yleisen ulkonäön vaihdellessa. Menetelmiä voidaan yhdistellä, jolloin voidaan saada paremmin erilaisia piirteitä kuvaavia tunnuslukuja. Tunnuslukuja voidaan myös käyttää syötteinä koneälyä hyödyntävälle luokittelijalle, kuten esimerkiksi neuroverkolle, joka suorittaa lopullisen piirteen tunnistuksen.

2.1 Gray Level Co-Occurrence Matrix - GLCM

Harmaasävykuvalle voidaan suorittaa tekstuurianalyysi hyödyntäen Haralick et al. esittämää harmaasävyjen yhteisesiintymismatriisia (GLCM). Matriisi luodaan tarkastelemalla harmaasävykuvassa esiintyvien pikseliparien määrää määrätyssä suunnassa. Lopputuloksena saadaan matriisi (Kuva 8), jossa on harmaasävytasojen neliön verran elementtejä, 8 bittisen kuvan tapauksessa matriisi olisi siis 256 x 256 elementin suuruinen.



Kuva 8. Esimerkki GLCM laskennasta, kun harmaasävyjä on 8 kappaletta ja vertailu suoritetaan oikeanpuoleiseen pikseliin.

Matriisista voidaan laskea ainakin 28 erilaista tunnuslukua, kuten yhtälöiden (8) ja (9) kontrasti sekä ASM (angular second moment) (Haralick et al. 1973):

$$\sum_{n=0}^{N_g-1} n^2 \left\{ \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \right\}_{|i-j|=n} \tag{8}$$

$$\sum_i \sum_j \{p(i,j)\}^2 \tag{9}$$

missä N_g on harmaasävyjen määrä kuvassa
i on pikselin vaakasuuntainen koordinaatti
j on pikselin pystysuuntainen koordinaatti
p_{ij} on pikselin intensiteettiarvo

2.2 Canny -reunantunnistusalgoritmi

Pitkien jatkuvien piirteiden, kuten halkeamien ja naarmujen, tunnistuksessa hyödyllistä on koittaa löytää piirteen reuna, joka väistämättä eroaa muista kuvassa näkyvistä reunoista pituudellaan ja jatkuvuudellaan. Reunantunnistukseen eräs hyvä menetelmä on Cannyn vuonna 1986 esittämä algoritmi. Menetelmä perustuu kuvasta laskettuihin gradientteihin. Reunan kohdalla gradientin voimakkuus on suurempi, sillä tyypillisesti reunan molemminpuolisten intensiteettiarvojen erotus on suuri. Algoritmissa ensin vähennetään kohinaa Gaussin konvoluution avulla, jonka jälkeen suoritetaan konvoluutio Gaussin konvoluutiomatriisin toisen suuntaderivaatan avulla yhtälön (10) lailla (Canny 1986):

$$\frac{\delta^2}{\delta n^2} G * I = 0 \tag{10}$$

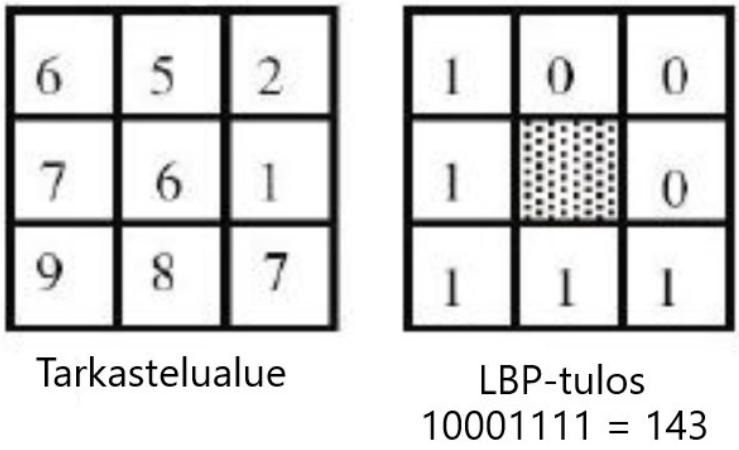
missä n on derivoinnin suunta
G on Gaussin konvoluutiomatriisi
I on kuva, jolle reunantunnistus suoritetaan
$*$ esittää konvoluutiota

Reunan vahvuutta voidaan arvioida yhtälön (11) avulla (Canny 1986):

$$|\nabla(G * I)| \tag{11}$$

2.3 Local Binary Pattern – LBP

Harmaasävykuvalle voidaan prosessitehokkaasti suorittaa tekstuurianalyysi Wang & He:n vuonna 1990 esittämiä paikallisia binäärikuvioita (LBP) hyödyntäen. Binäärikuviot voivat kertoa myös suoraan, onko tarkasteltava alue nurkka tai reuna. Menetelmässä tarkastellaan, mitkä pikselit tarkastelupikselin ympärillä ovat suurempia tai yhtä suuria kuin tarkastelupikseli. Tarkastelupikseliä suuremmat tai yhtä suuret ympäröivät pikselit asetetaan konseptuaalisesti ”päälle”. Tuloksena saadaan 8 bittinen luku, joka kuvaa pikselin ympäristöä (Kuva 9). Menetelmä on tehokas, sillä se perustuu lukujen vertailulle, eikä laskutoimituksille.

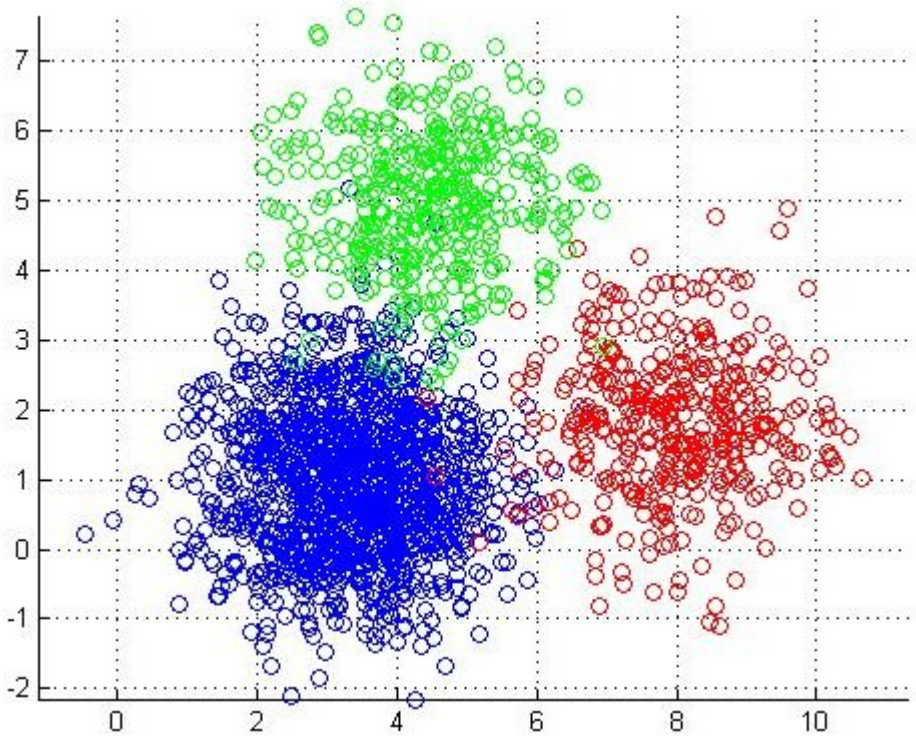


Kuva 9. Esimerkki LBP-laskennasta.

3 VIRHEEN LUOKITTELUMENETELMÄT

Kun kuvasta on laskettu erilaisia tunnuslukuja, voidaan lukuja hyödyntää tunnistamaan kuvassa olevia piirteitä. Esimerkiksi kuvalle, jolle on suoritettu reunantunnistus, voidaan halutut piirteet (esimerkiksi halkeamat) löytää käymällä kuva pikseli pikseliltä läpi etsien pitkiä yhtenäisiä alueita. Tällaiset menetelmät ovat kuitenkin alttiita virheille.

Tarkoituksenmukaisempaa on käyttää menetelmiä, jotka hyödyntävät tunnetuista piirteistä tiedettyjä arvoja ja etsivät näiden avulla uusista kuvista vastaavia piirteitä. Klusteroinnissa tunnuslukujen arvojoukosta pyritään löytämään yhtenäisiä alueita, klustereita. Käytännössä tämä tapahtuu etsimällä arvot, joiden etäisyys toisistaan on lyhyin mahdollinen. Kunhan eri piirteiden tuottamat arvot eriävät toisistaan tarpeeksi, asettuvat arvot klustereiksi (Kuva 10). Uuden kuvan arvoja voi tämän jälkeen verrata eri klustereihin ja luokitella kuva yhteensopivimman klusterin perusteella. (Shapiro & Stockman 2001, s. 307)



Kuva 10. Esimerkki klusteroinnista.

3.1 K-Means

K:n mediaanin menetelmässä (K-Means) klustereita oletetaan olevan määrä K, jolloin jokaisella klusterilla on oma mediaaniarvonsa. Menetelmässä voidaan joko arvata klustereille mediaaniarvo, jonka jälkeen arvauksen tarkkuus tarkistetaan laskemalla arvojen etäisyydet mediaanista, tai arvot voidaan asettaa mielivaltaisiin klustereihin, jonka jälkeen lasketaan joukkojen mediaanit ja tarkistetaan arvojen etäisyydet niistä. Arvojen etäisyys mediaanista pyritään minimoimaan, jolloin saadaan tiivein mahdollinen klusteri. Klusterin tiiveyttä voidaan arvioida yhtälön (12) avulla (Shapiro & Stockman 2001, s. 308):

$$D = \sum_{k=1}^{K} \sum_{x_i \in C_k} \left|\left|x_i - m_k\right|\right|^2 \tag{12}$$

missä D kuvaa arvojen läheisyyttä mediaaniin

x_i on tarkasteltava arvo

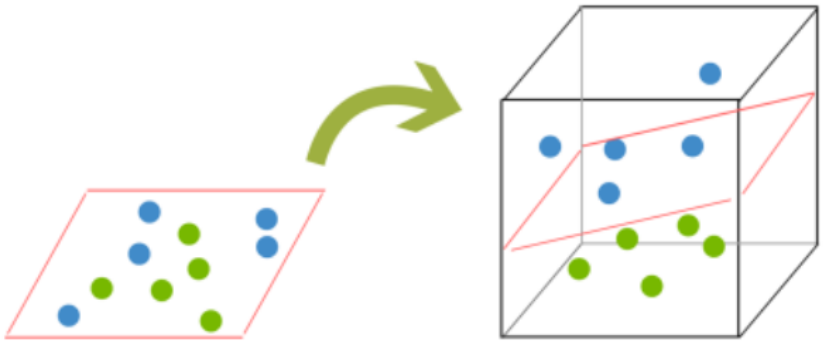
m_k on tarkasteltava mediaani

3.1.1 K-Nearest Neighbor – KNN

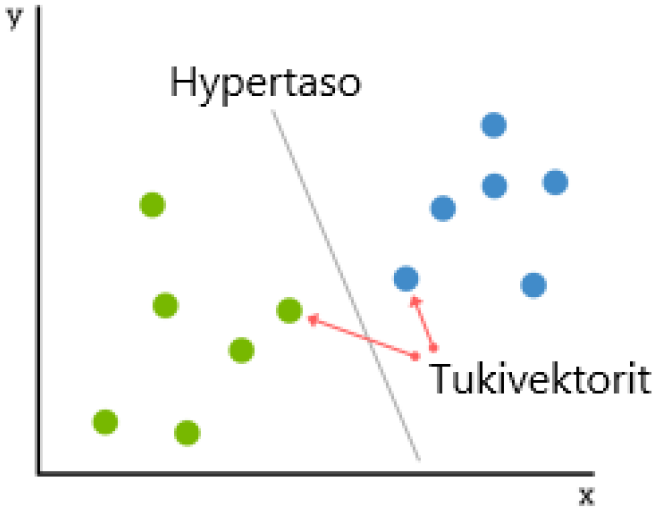
K:n lähimmän naapurin menetelmässä (KNN) tarkastellaan mediaanien sijaan arvojen etäisyyttä toisistaan. Klustereissa pyritään tällöin minimoimaan arvojen kollektiivinen etäisyys toisistaan.

3.2 Support Vector Machine - SVM

Tukivektorikone (SVM) pyrkii erottamaan arvojoukot toisistaan hypertasolla. Koska arvojoukot voivat olla sekoittuneet toisiinsa, kone koittaa sovittaa tasoa korkeammissa dimensioissa (Kuva 11). Koneen tarkoituksena on koittaa löytää taso, joka on mahdollisimman kaukana arvoista, kuitenkin jakaen ne toisistaan erilleen. Kone hyödyntää tähän tukivektoreita, jotka ovat tasoa lähimmät arvopisteet (Kuva 12). (Cortes & Vapnik 1995)



Kuva 11. Esimerkki tukivektorikoneen hypertason sovittamisesta korkeammassa dimensiossa.



Kuva 12. Esimerkki tukivektoreista.

3.3 Syväoppiminen

Tässä työssä tarkasteltavassa OpenCV -konenäkökirjastossa ei ole kattavia syväoppimisen ominaisuuksia, mutta aihe on syytä silti mainita sen kasvavan suosion ja laajojen käyttökohteiden takia. Tensorflow ja sen pohjalle rakennettu Keras, sekä PyTorch Python kirjastot tarjoavat hyviä syväoppimisen työkaluja.

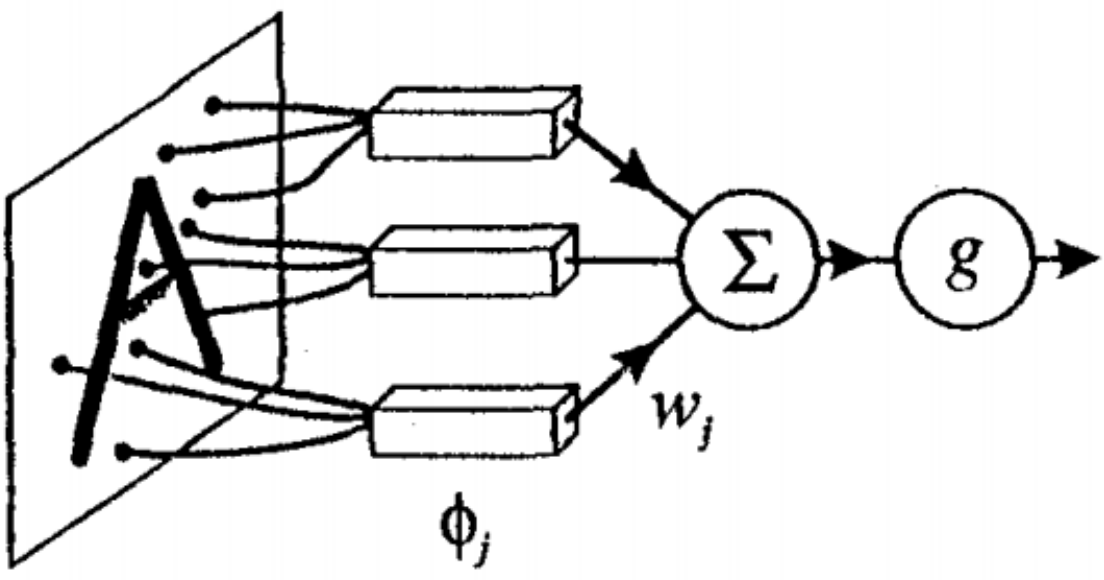
Syväoppimisessa hyödynnetään useita epälineaarisia prosessointikerroksia ja alempitasoisten piirteiden pohjalta laskettuja korkeampitasoisia piirteitä (Deng & Yu 2014, s. 199–200), joka erottaa sen selvästi aiemmin mainituista menetelmistä, joissa tehdyt toimenpiteet olivat lineaarisia ja malli hyödynsi vain hyvin matalatasoisia piirteitä, jolloin mallin opetus sekä johtopäätökset voidaan kartoittaa ja ne ovat helposti ymmärrettävissä. ”Syvä” nimike juontuu monikerroksisesta rakenteesta.

3.3.1 Neuroverkko

”Neuroverkko” nimitys pohjautuu pyrkimyksiin koittaa matemaattisesti jäljitellä biologisia järjestelmiä. Kuvantunnistuksen tarpeisiin biologisten järjestelmien realistinen jäljittely tuottaisi kuitenkin turhia rajoitteita, joten tilalle on kehitelty tehokkaampia verkkoja, jotka ovat merkittävästi yksinkertaisempia. (Bishop 2006, s. 226)

Neuroverkko koostuu perseptroneista. Perseptroni koostuu prosessointielementeistä (ϕ_j), adaptiivisista painoista (w_j) ja aktivointifunktiosta (g) (Kuva 13). Perseptronille syötetään dataa, josta se tuottaa piirrevektoreita prosessointielementtien ja painojen avulla. Tämän jälkeen aktivointifunktio päättää piirrevektoreiden arvoja katsomalla, mihin luokkaan tarkasteltu data kuuluu. Perseptronin opetus tapahtuu syöttämällä dataa ja tarkastamalla perseptronin tuottaman luokittelun oikeus. Jos luokittelu meni oikein, mitään ei tehdä. Jos luokittelu meni väärin, painoja muutetaan jollain halutulla menetelmällä. (Bishop 1995, s. 98–100)

Perseptroneja voidaan ketjuttaa, jolloin saadaan korkeampitasoisia piirteitä, joiden pohjalta kouluttaa mallia. Piirteet riippuvat valituista prosessointielementeistä, kuvantunnistuksessa elementit voivat suorittaa esimerkiksi konvoluutioita.



Kuva 13. Esimerkki perseptronista (Bishop 1995).

4 OPENCV JA ESIMERKKEJÄ

OpenCV (Open Computer Vision) on avoimen lähdekoodin konenäkökirjasto, joka sisältää kuvien käsittelyä helpottavia työkaluja sekä työkaluja koneoppimisen tarpeisiin. Näihin lukeutuu yli 2500 optimoitua algoritmia. Kirjasto tukee Python, C++, Java ja MATLAB -ohjelmointikieliä sekä Windows, Mac OS, Linux ja Android -käyttöjärjestelmiä. Kirjasto on C++-pohjainen ja suunnattu reaaliaikaisen konenäön tarpeisiin. Kehitteillä on myös tuki grafiikkakiihdytteiselle prosessoinnille hyödyntäen CUDA ja OpenCL käyttöliittymiä. (OpenCV 2021)

OpenCV:llä on modulaarinen rakenne, joka koostuu useista jaetuista kirjastoista. Päärakenne on seuraavaa muotoa (OpenCV 2021):

- Ydintoiminnot (core), määrittelee datarakenteet ja perusfunktiot, kuten kuvien lataamisen ja tallentamisen.
- Kuvankäsittely (imgproc), sisältää työkaluja lineaariseen ja epälineaariseen kuvan suodatukseen, kuvien geometriseen muunteluun, väriavaruuden muuntamiseen, histogrammien tekoon yms.
- Videoanalyysi (video), sisältää työkaluja liikkeen arviointiin, taustan poistoon ja esineen seurantaan.
- Kameroiden kalibrointi ja 3D rekonstruktio (calib3d), sisältää työkaluja yksittäis- ja stereokameroiden kalibrointiin, esineen asennon arviointiin, elementtejä 3D rekonstruktioon yms.
- 2D piirteet (features2d), sisältää työkaluja keskeisten piirteiden laskemiseen sekä kuvaajia ja kuvaajien yhdistelijöitä.
- Esineen tunnistus (objdetect), sisältää työkaluja esimääriteltyjen luokkien tunnistamiseen (kuten kasvot, silmät, autot yms.).
- Korkean tason käyttöliittymä (highgui), sisältää työkaluja yksinkertaisen käyttöliittymän luontiin.
- Video I/O (videoio), helppokäyttöinen käyttöliittymä videoiden ottamiseen ja video koodekkeihin.

Näiden moduulien lisäksi kirjastossa on 64 muutakin moduulia.

OpenCV hyödyntää NumPy-matriiseja, joita on huomattavasti helpompi käsitellä verrattuna esimerkiksi Pythonin omiin ratkaisuihin (sisäkkäiset listat). Kuvan lataaminen ja rajaus toimii OpenCV:ssä esimerkiksi näin (Python):

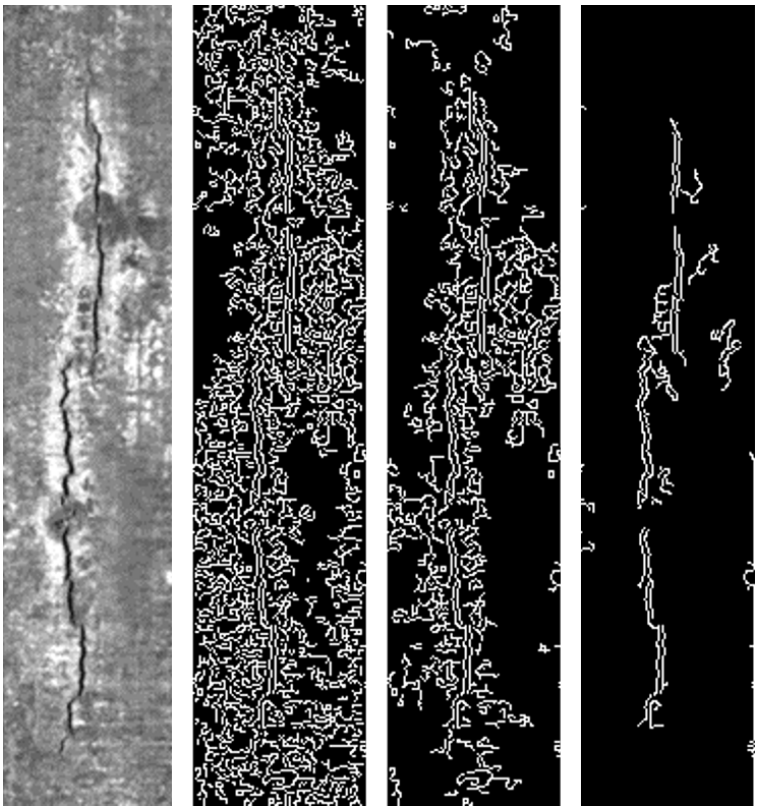
```
import cv2 as cv

imgsrc = "kuvanpolku/kuva.jpg"
img = cv.imread(imgsrc, cv.IMREAD_GRAYSCALE)

img = img[100:500, 100:500]
```

Tuloksena `img` muuttujassa on NumPy matriisina 400x400 pikselin kuva. `cv.IMREAD_GRAYSCALE` argumentti määrää, että kuva asetetaan muuttujaan harmaasävykuvana.

Yksi tapa lähestyä esimerkiksi halkeamien löytämistä valuaihiosta, on tunnistaa halkeaman reuna. Reuna eroaa muista reunoista aihiossa pituudellaan, vahvuudellaan ja jatkuvuudellaan. OpenCV:ssä kuvalle voidaan Cannyn algoritmilla suorittaa reunantunnistus näin:

```
import cv2 as cv

src = "kuva.png"
img = cv.imread(src, cv.IMREAD_GRAYSCALE)

low_threshold = 220
upper_threshold = low_threshold * 3

canny = cv.Canny(img, low_threshold, upper_threshold)

cv.imwrite("canny.png", canny)
```

`low_threshold` ja `upper_threshold` määrittelee, minkä vahvuiset reunat huomioidaan (Kuva 14). Lopuksi kuva tallennetaan ”canny.png” nimisenä tiedostona.



Kuva 14. Esimerkki halkeamalle tehdystä reunantunnistuksesta low_threshold ollessa 60, 120 ja 220.

Kuten tuloksesta nähdään, halkeama erottuu selvästi, kunhan raja-arvot ovat oikein. Tämän jälkeen reunatunnistetusta kuvasta voidaan jollakin menetelmä etsiä pitkää yhtenäistä aluetta automaattisen tunnistuksen mahdollistamiseksi. Ongelmallista tässä menetelmässä on sen virheensietokyky. Jos pinnan ulkonäkö vaihtelee, halkeaman reunan tunnistettavuus vaihtelee eikä parametrien säätö välttämättä riitä kattamaan tuota vaihtelua. Pinnan kohinaa, joka näkyy esimerkissä etenkin reunan vahvuuden alarajan ollessa 60, voidaan vähentää esikäsittelemällä kuvaa esimerkiksi tasoittamalla sitä. Tasoitusmenetelmä on syytä valita niin, ettei se vaikuta halkeaman reunan tyyppisiin alueisiin. Tämänkin toiminnalle on kuitenkin rajansa pinnan vaihdellessa paljon.

4.1 GLCM-tarkastelu

Reunantunnistuksen tyyppinen operaatio on teräspintojen kohdalla melko virhealtis, eikä ole hyödynnettävissä kovin monen virhetyypin tunnistamiseen. Tarkoituksenmukaisempaa on löytää jokin yleistävä tunnusluku, jossa erityyppiset virheet erottuvat. Aiemmin mainitut GLCM ja LBP voisivat olla tällaisia. OpenCV

kirjastosta ei kuitenkaan toistaiseksi löydy valmiiksi näitä algoritmeja. Algoritmit voitaisiin luoda itse OpenCV:tä apuna käyttäen, mutta koska valmiita ratkaisujakin löytyy, olisi tämä melko turhaa. Käytetään siis skimage-kirjastosta löytyviä algoritmeja. Kokeeksi luodaan ohjelma, joka laskee kuvasta harmaasävyjen yhteisesiintymismatriisin ja tästä matriisista muutaman tunnusluvun.

```
from skimage.feature import greycomatrix, greycoprops
import cv2 as cv

src = "kuva.png"
img = cv.imread(src,cv.IMREAD_GRAYSCALE)

glcmmat = greycomatrix(img, [1], [0])
con = greycoprops(glcmmat, "contrast")
hgen = greycoprops(glcmmat, "homogeneity")
asm = greycoprops(glcmmat, "ASM")
corr = greycoprops(glcmmat, "correlation")
diss = greycoprops(glcmmat, "dissimilarity")
```

`greycomatrix` funktiolle annetaan argumentteina tarkasteltavien pikseleiden etäisyys toisistaan ja tarkastelun suunta. Tässä tapauksessa tarkastellaan heti tarkastelupikselin viereistä pikseliä, joten arvo on 1. Tarkastelun suunnaksi on asetettu 0, jolloin tarkastellaan tarkastelupikselin oikeanpuoleista pikseliä.

Vertaillaan lopuksi tunnuslukujen arvoja ”puhtaan” pinnan ja halkeaman välillä. Arvot laskettiin 75 esimerkille puhtaasta pinnasta ja 13 esimerkille halkeamasta (Taulukko 1).

Taulukko 1. GLCM-tunnuslukujen keskiarvot halkeamille ja puhtaalle pinnalle.

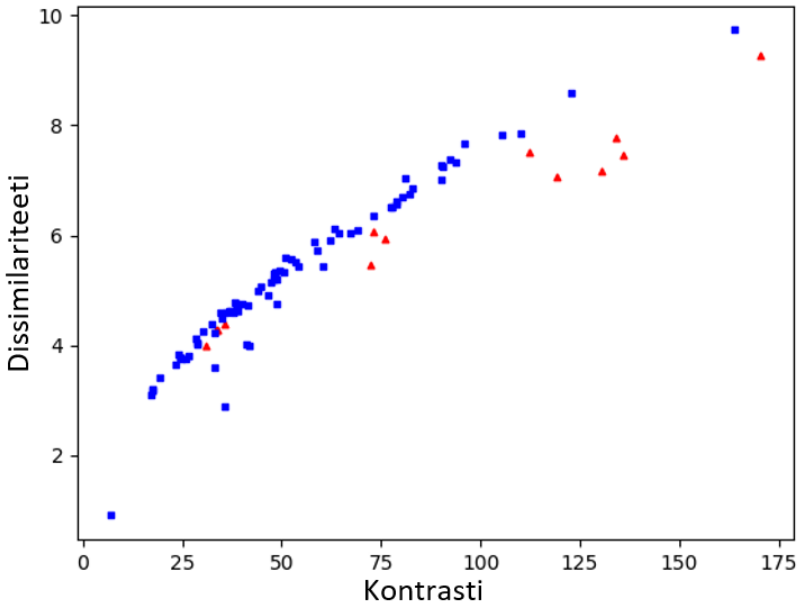
	Kontrasti	Homogeenisyys	ASM	Korrelaatio	Dissimilariteetti
Halkeamat	89.10091605	0.186129579	0.000885726	0.931943986	6.206713301
Puhdas	53.63596409	0.210863435	0.00648154	0.927739157	5.275386756

Huomataan, etteivät arvot juurikaan eroa toisistaan. Merkittävin ero on ASM-arvon kohdalla, jossa puhtaalla pinnalla on suhteessa huomattavasti suurempi arvo. Tämä käy järkeen, sillä ASM kuvastaa kuvan ”energiaa”, eli intensiteettiparien neliösummaa (Kaava 9). Käytännössä numero on siis sitä isompi, mitä enemmän kuvassa on yhden tyyppisiä pikselipareja. Puhdas pinta on tyypillisesti melko jatkuvaa eikä suuria eroja ole, joten tiettyjä pikselipareja on enemmän. Kokeillaan vaihtaa tarkastelun suuntaa, jos tällä olisi vaikutusta lukujen eroihin (Taulukko 2).

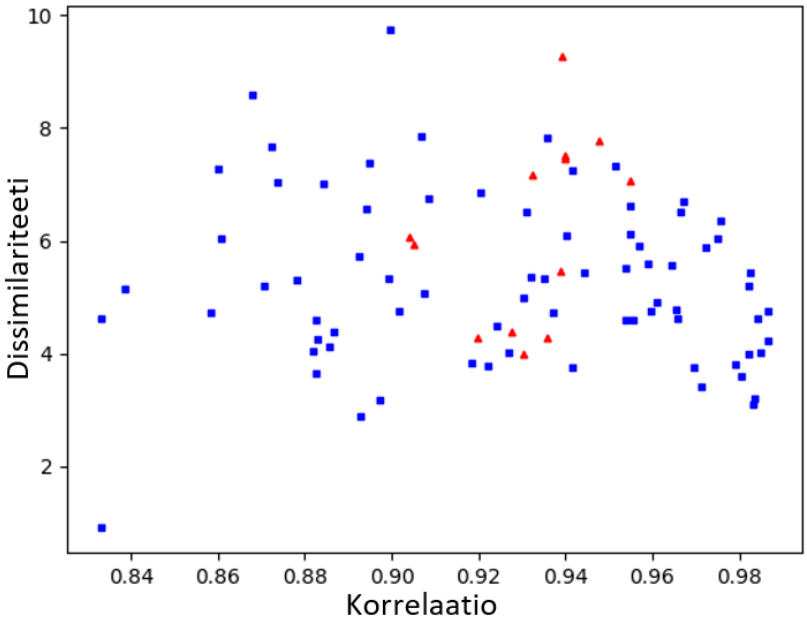
Taulukko 2. GLCM-tunnuslukujen keskiarvot halkeamille ja puhtaalle pinnalle tarkastelun suunnan ollessa 90 astetta.

	Kontrasti	Homogeenisyys	ASM	Korrelaatio	Dissimilariteetti
Halkeamat	60.49744334	0.196720765	0.000921447	0.946806914	5.460044209
Puhdas	43.89001159	0.22550142	0.006621658	0.941623753	4.781478261

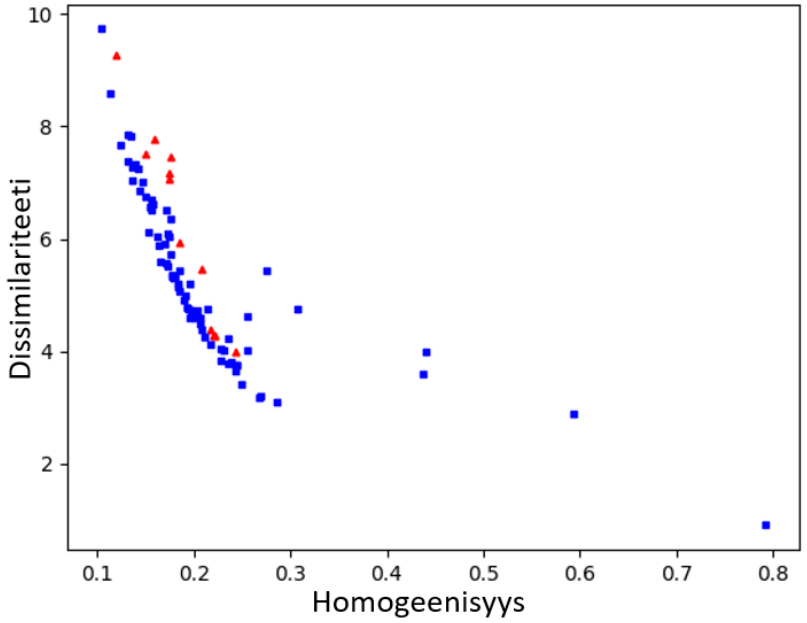
Jälleen erot eivät ole suuria ja suurin suhteellinen eroavaisuus näkyy ASM-arvossa. Kuvista 15, 16 ja 17 nähdään, etteivät arvot eroa merkittävästi toisistaan. Luokittelu näiden tunnuslukujen pohjalta on siis hankalaa.



Kuva 15. Kuvaaja dissimilariteetin ja kontrastin arvoista, suunta 0, punainen on halkeama.



Kuva 16. Kuvaaja korrelaation ja dissimilariteetin arvoista.



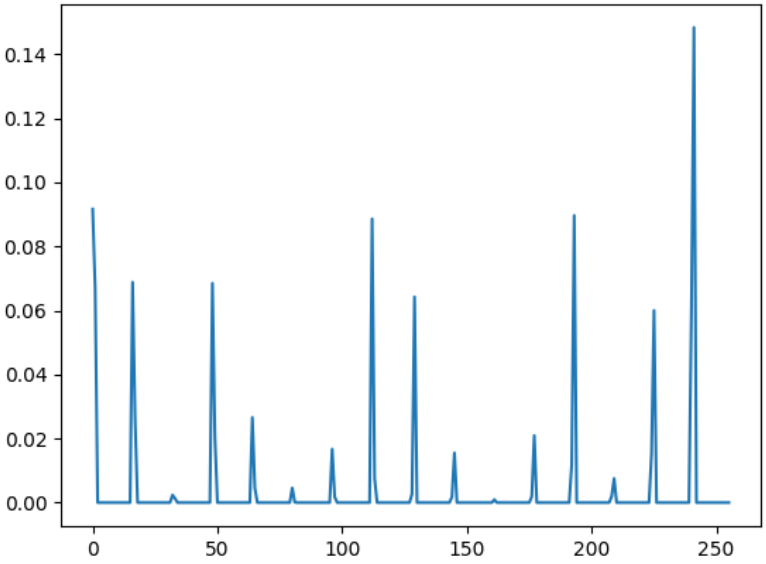
Kuva 17. Kuvaaja homogeenisyyden ja dissimilariteetin arvoista.

4.2 LBP-tarkastelu

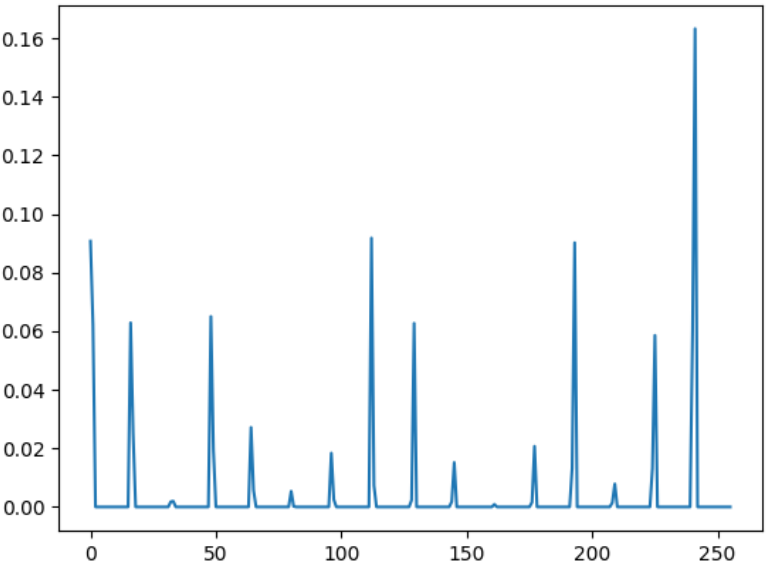
Kokeillaan laskea kuvasta LBP-kuva, ja luoda tästä kuvasta histogrammi. Histogrammista pitäisi kyetä näkemään kuvan sisällöstä riippuen erityyppisiä tekstuurillisia muotoja.

```
from skimage.feature import local_binary_pattern
import cv2 as cv
import numpy as np

histSize = 256
histRange = (0, 255)

masterhist = np.zeros((histSize, 1))

for src in srcs:
    img = cv.imread(src,cv.IMREAD_GRAYSCALE)
    lbpimg = local_binary_pattern(img, 8, 1)
    hist = cv.calcHist(np.uint8(lbpimg), [0], None, [histSize], histRange)
    masterhist = np.add(masterhist,hist)

masterhist = np.true_divide(masterhist, np.sum(masterhist))
```

`local_binary_pattern` funktiolle annetaan argumentteina LBP matriisin “säde” sekä tarkasteltavien pikselien määrä kehällä. Histogrammin laskemiseen hyödynnetään OpenCV:n `calcHist` funktiota, jolle kerrotaan argumentteina histogrammin haluttu koko. On haluttavampaa luoda histogrammi, joka kuvaa useita tietyn tyypin kuvia, joten ohjelmassa luodaan `masterhist` niminen histogrammi, johon kaikkien tarkasteltujen kuvien histogrammit ynnätään yhteen. Histogrammin arvot skaalataan jakamalla jokaisen histogrammin ”lokeron” arvo histogrammin arvojen kokonaismäärällä, eli käytännössä LBP kuvien pikselien määrällä. Tällöin histogrammi kertoo eri arvojen osuudet kuvissa (Kuva 18 & Kuva 19). Histogrammeissa on eroa etenkin viimeisen piikin suuruudessa, mutta yllättävää on, että histogrammien yleinen muoto vaikuttaa identtiselle. Tämä on oikeastaan odotettava tulos, sillä LBP on nimenomaan tekstuurianalyysiin soveltuva työkalu. Koska halkeama on melko pieni osa kuvasta, korostuu histogrammissa kuvan yleinen tekstuuri, joka vastaa karkeasti puhtaan pinnan tekstuuria. Näennäisesti siis tämäkään menetelmä ei sovellu, ainakaan halkeamien, tunnistamiseen erityisen hyvin.

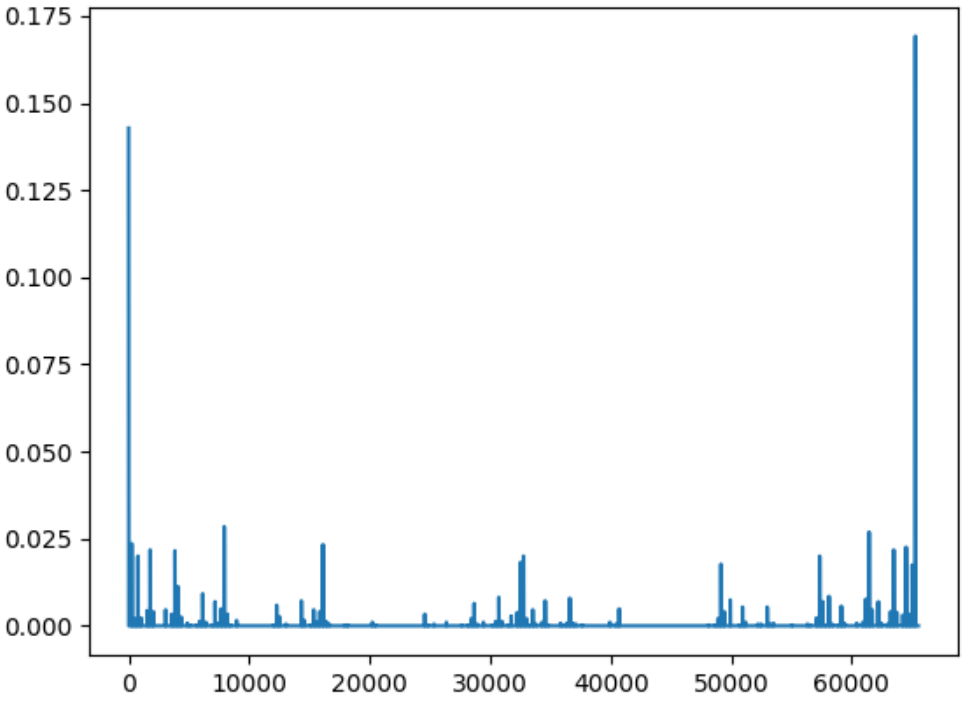


Kuva 18. Puhtaan pinnan skaalattu LBP-histogrammi.

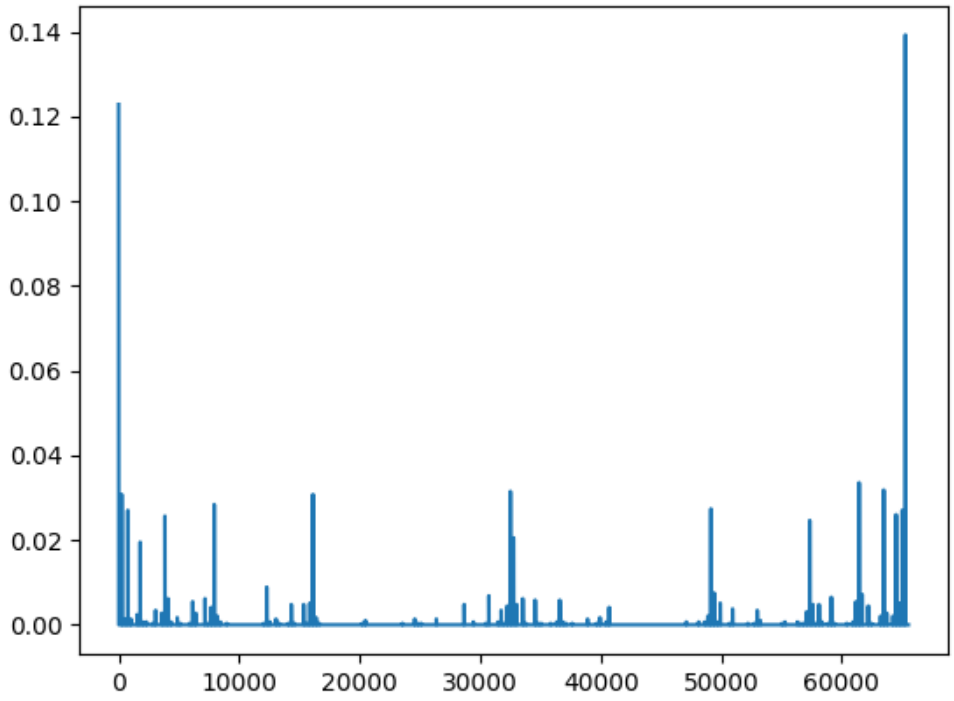


Kuva 19. Halkeamien skaalattu LBP-histogrammi.

Kokeillaan vielä laskea LBP säteellä 2, jolloin matriisin kehällä on 16 pikseliä. Tällöin histogrammissa on 2^16 kappaletta lokeroita ja pikselin luokittelun vaihtoehdot kasvavat (Kuva 20 & Kuva 21). Toisaalta tarkasteltavaa pikseliä lähimpiä pikseleitä ei tällöin huomioida lainkaan. Huomataan, etteivät histogrammit eroa tälläkään tavalla merkittävästi toisistaan. Toisaalta ensimmäinen ja viimeinen piikki voitaisiin jättää huomioitta, jolloin saatettaisiin nähdä merkittävämpiä eroja histogrammien pienemmissä piikeissä.



Kuva 20. Puhtaan pinnan skaalattu 16-bittinen LBP-histogrammi.



Kuva 21. Halkeamien skaalattu 16-bittinen LBP-histogrammi.

4.3 Tarkastelu OpenCV:n deskriptoreilla

Puhtaasti OpenCV:tä hyödyntäviä tunnistusmenetelmiä etsiessä kokeiltiin OpenCV:stä löytyviä piirredetektoreita ja deskriptoreita, kuten ORB (Oriented FAST and Rotated BRIEF) ja SIFT (Scale Invariant Feature Transform). Nämä etsivät kuvasta kiinnostavia piirteitä ja laskevat näiden pohjalta deskriptorivektoreita. Nämä vektorit klusteroitiin käyttäen K-Meansiä, jolloin saatiin kuvan mediaanivektori.

```
orb = cv.ORB_create()
_, des = orb.detectAndCompute(img, None)
_, _, center = cv.kmeans(np.float32(des), 1, None,
(cv.TERM_CRITERIA_MAX_ITER, 10, 1), 50, cv.KMEANS_RANDOM_CENTERS)
```

Kun tämä prosessi suoritettiin Sapotech Oy:n tarjoamalle kuvakannalle, saatiin tuloksena suuri määrä halkeamia ja puhdasta pintaa kuvastavia vektoreita. Näillä vektoreilla opetettiin SVM-malli.

```
svm = cv.ml.SVM_create()
svm.setType(cv.ml.SVM_NU_SVC)
svm.setNu(0.01)
svm.setKernel(cv.ml.SVM_RBF)
svm.setTermCriteria((cv.TERM_CRITERIA_MAX_ITER, 10000, 1))
svm.train(trainingData, cv.ml.ROW_SAMPLE, labels)
```

Lopputuloksena eri deskriptoreita testaamalla sekä SVM-mallin hyperparametreja säätämällä voitiin parhaimmillaan saavuttaa erillistä validointidataa vasten taulukon 3. mukaiset ennustukset. Ennustuksen ollessa väärin malli ennustaa halkeaman puhtaana pintana ja puhtaan pinnan halkeamana.

Taulukko 3. Validointidatalle tehdyt ennustukset.

	Oikein	Väärin
Halkeamat	78	60
Puhdas	1553	607

5 YHTEENVETO

Konenäössä lähestymistapoja on monia, mutta suurin rajoittava tekijä kaikilla tavoilla on kuvien yleinen luonne ja digitaalisen kuvan rakenne. Kuvan tunnistaminen koneellisesti ei läheskään aina ole intuitiivinen prosessi. Päinvastoin asia, joka on ihmiselle todella helppo tehtävä, on koneellisesti todella hankalaa, ellei joissain tapauksissa jopa mahdotonta toteuttaa.

LBP ja GLCM -menetelmät eivät tuottaneet tarpeeksi kuvaavia tunnuslukuja ainakaan halkeaman tunnistamiseen teräspinnasta. Tämä on toisaalta odotettava tulos, sillä nämä menetelmät ovat kehitetty tekstuurianalyysiin kiinnostavien piirteiden löytämisen sijaan. Cannyn reunantunnistus algoritmissa on selvää potentiaalia yhdeksi halkeaman tunnistusvaihtoehdoksi, mutta se tuskin tulee toimimaan yksinään tähän tehtävään pinnanvaihtelusta syntyvän virhealttiuden takia.

OpenCV:n deskriptorit toimivat odotettua paremmin halkeaman tunnistukseen. Useat kirjastosta löytyvät deskriptorit soveltuvat paremmin orgaanisten tai todella keinotekoisten piirteiden tunnistukseen. Teräspinnat jäävät ikävälle harmaalle alueelle vaihtelevuutensa takia. Kaikki halkeamaa esittävästä kuvasta lasketut deskriptorit eivät välttämättä ole itse halkeaman alueesta laskettuja, vaan ne voivat myös kuvata halkeaman ympäristöä. Halkeaman ympäristö on ulkonäöltään hyvin lähellä puhdasta teräspintaa, joten tämä voi sekoittaa kuvasta laskettavaa mediaanivektoria.

SVM-mallin hyödyntäminen lopulliseen luokitteluun esittää myös monta muuttujaa. Jos mallille syötettävässä datassa on paljon päällekkäisyyttä eri luokkien välillä, on hypertason sovittaminen luokkien väliin hankalaa ja malli joutuu tekemään kompromisseja. Tällöin ennustustarkkuus laskee.

Kuten esimerkeissä huomataan, ei teräspintojen virheiden tunnistus ole helpoin mahdollinen tehtävä edes halkeaman kaltaisen selvän virheen kohdalla. Koneoppimista hyödyntääkseen on löydettävä hyvin kuvaava tunnusluku, joka toimii myös virheen luonteen ja sen potentiaalisen esiintymisympäristön vaihdellessa. Joissain tapauksissa helpompaa voikin olla korvata koneoppiminen vaiheittaisella, useita algoritmeja hyödyntävällä ”manuaalisella” menetelmällä. Prosessitehokkuutta on myös syytä

tarkastella etenkin reaaliaikaisia tunnistusjärjestelmiä kehitettäessä. Datan luonteesta riippuen kaikki luokittelumenetelmät eivät välttämättä myöskään toimi ongelmitta. Jos valittu tunnusluku ei erota eri piirteitä toisistaan tarpeeksi selvästi, on luokittelu hankalaa menetelmästä riippumatta.

Tunnistamisen ongelmat voivat juontua myös kuvasta laskettujen piirteiden matalatasoisuuteen. Klusteroinnin kaltaisille menetelmille ”piirretasojen” lisääminen on hankalaa, sillä menetelmissä ei ole keinoa yhdistää useita tasoja toisiinsa halutulla tavalla. Neuroverkot voisivat olla ratkaisu tähän ongelmaan, jolloin kuvasta saataisiin hyödynnettyä merkittävästi suurempi määrä tietoa monitasoisuuden kautta.

LÄHDELUETTELO

Bishop, C. M., 1995. Neural networks for pattern recognition. Oxford: Clarendon. ISBN: 0-19-853864-2

Bishop, C. M., 2006. Pattern Recognition and Machine Learning. Springer. ISBN: 978-0-387-31073-2

Canny, J., 1986. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 8(6) s. 679-698.

Cortes, C., & Vapnik, V., 1995. Support-vector networks. Machine Learning, 20(3) s. 273-297.

Davies E. R., 2005. Machine Vision: Theory, Algorithms, Practicalities. Amsterdam: Morgan Kaufmann. ISBN: 978-0-12206-093-9

Deng, L. & Yu, D., 2014. Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing. 7(3–4) s. 197-387.

Gonzalez, R., and Woods, R., 2018. Digital Image Processing, Global Edition, 4th Edition. New York: Pearson International Content. ISBN: 978-1-29222-304-9

Haralick, R. M., Shanmugam, K., Dinstein, I., 1973. Textural Features for Image Classification. IEEE Transactions on Systems, Man, and Cybernetics, SMC-3(6) s. 610-621.

Moore, G. E., 1965 Cramming more components onto integrated circuits. Electronics Magazine. 38(8) s. 114–117.

OpenCV, 2021. [verkkodokumentti] Palo Alto, CA: OpenCV. Saatavissa: https://opencv.org/ [viitattu 30.6.2021]

Shapiro, L. G. & Stockman, G. C. 2001 Computer vision. Upper Saddle River (N.J.): Prentice Hall. ISBN: 0-13-030796-3

Wang, L., & He, D., 1990. A new statistical approach for texture analysis. 56(1) s. 61-66.